
A Hybrid Monte Carlo Architecture for Parameter Optimization

May 13, 2014

Author: James Brofos
 james.a.brofos.15@dartmouth.edu
 Dartmouth College

Advisor: Meifang Chu
 Lecturer in Mathematical Finance
 Dartmouth College

Abstract

Much recent research has been conducted in the area of Bayesian learning, particularly with regard to the optimization of hyper-parameters via Gaussian process regression [1, 2]. The methodologies rely chiefly on the method of maximizing the expected improvement of a score function with respect to adjustments in the hyper-parameters. In this work, we present a novel algorithm that exploits notions of confidence intervals and uncertainties to enable the discovery of the best optimal within a targeted region of the parameter space. We demonstrate the efficacy of our algorithm with respect to machine learning problems and show cases where our algorithm is competitive with the method of maximizing expected improvement.

1 Introduction

We begin by formally defining the notion of the Gaussian process as prior distribution on functions f where $f : \Theta \rightarrow \mathbb{R}$ and we consider Θ to be a “state-space” of parameters. In particular, given tuples $(\theta, y) \in \Theta \times \mathbb{R}$, we assume that $y \sim \mathcal{N}(f(\theta), \sigma^2)$. We say that a series of such tuples, of cardinality n , induces a multivariate Gaussian distribution in \mathbb{R}^n . This Gaussian architecture is appealing for several reasons:

1. It elegantly fits a basis function to the data, allowing for trivial inference of the behavior of all points in Θ .
2. The underlying Gaussian assumptions permit statistical notions of *expected improvement* and *uncertainty* to arise in closed form from the fitted model.
3. The prior two points lead naturally to a framework that enables Gaussian processes to optimize parameters in machine learning models via a principled search of Θ .

Optimization frameworks of this form offer an immediate advantage over discrete parameter optimization methodologies such as k -fold cross-validation, which requires k^m performance evaluations of the learned model if m is the cardinality of the discrete set. This is computationally expensive and fails to generate knowledge of the model’s performance for $\theta \in \Theta$ when θ is not a member of the discrete parameter set used by cross-validation.

Current generation Gaussian process optimization methods exploit the the expected improvement of the model performance above the current best at all points in Θ . The improvement at θ^* is $\mathcal{I}(\theta^*) = f(\theta^*) - y_{\text{best}}$, where y_{best} is the current best score of the objective function. It can be shown [3] that the expected improvement is:

$$\mathbb{E}[\mathcal{I}(\theta^*)] = \max\{0, \sigma(\theta^*)[u\Phi(u) + \phi(u)]\} \quad (1)$$

Here we represent the standard deviation of the prediction at θ^* as $\sigma(\theta^*)$ and let $u = \frac{f(\theta^*) - y_{\text{best}}}{\sigma(\theta^*)}$. We also denote the CDF of the standard normal distribution as $\Phi(\cdot)$ and similarly for the standard normal PDF, $\phi(\cdot)$. The essential idea of these optimization algorithms is to pursue function evaluations at those points yielding highest expected improvement in the objective function, thereby extracting more information about the nature of the true, underlying objective function itself. This process is continued until no further function evaluations are expected to yield improvements.

In this work we consider additionally the applications of the *probability of improvement* to enhancing the Monte Carlo nature of our algorithm. The probability of improvement can be derived as:

$$\mathbb{P}[y_{\theta^*} > y_{\text{best}}] = \mathbb{P}\left[X < \frac{f(\theta^*) - y_{\text{best}}}{\sigma(\theta^*)}\right] \quad (2)$$

$$= \mathbb{P}[X < u] = \Phi(u) \quad (3)$$

Here we say that $X \sim \mathcal{N}(0, 1)$, which is trivially shown as true. For the purposes of this work, we

will refer to the expected improvement, probability of improvement, and mean-value criteria for point selection as *anticipation equations*.

1.1 Squared Exponential Covariance Function

Equally necessary to the definition of the Gaussian process is the covariance kernel, which permits the Gaussian process to express a versatile set of basis functions to fit the underlying objective function. A common choice of kernel is *squared exponential*, which defines a matrix \mathbf{C} :

$$\mathbf{C}(\theta_i, \theta_j) = \alpha \exp \left[-\frac{1}{2} \sum_{d=1}^D \frac{(\theta_i^{(d)} - \theta_j^{(d)})^2}{2\gamma_d^2} \right], \quad (4)$$

and a vector $\mathbf{k} = \mathbf{C}(\theta, \theta_i)$. The hyper-parameters $\{\alpha, \gamma_1, \dots, \gamma_D\}$ of the Gaussian process may be learned via maximum likelihood estimation by maximizing the *evidence* of the fitted model. These are functionally related to the prediction and variance of the prediction as follows:

$$f(\theta) = \mathbf{k}^T \mathbf{C}^{-1} \mathbf{y} \quad (5)$$

$$\sigma^2(\theta) = \mathbf{C}(\theta, \theta) - \mathbf{k}^T \mathbf{C}^{-1} \mathbf{k} \quad (6)$$

The squared exponential kernel is a frequent choice within the Gaussian process literature, so we select it here as a practical (and interpretable) baseline for our proposed methodology. Some authors have criticized this choice of kernel as providing an unreasonably smooth interpolation of the basis [1]. The alternative option is that of Snook et al. though we do not implement the Matérn $_{\frac{5}{2}}$ kernel:

$$\mathbf{C}(\theta_i, \theta_j) = \alpha \left(1 + \sqrt{5\Gamma} + \frac{5}{3}\Gamma \right) \exp(-\sqrt{5\Gamma}) \quad (7)$$

$$\Gamma = \Gamma(\theta_i, \theta_j) = \sum_{d=1}^D \frac{(\theta_i^{(d)} - \theta_j^{(d)})^2}{2\gamma_d^2} \quad (8)$$

2 Machine Learning Problem Formalism

In the context of machine learning with are typically presented with a model M , which is a function of the observations \vec{x}_i , the targets y_i , and the model parameters θ . The efficacy of this model can then be evaluated by a score function $\Psi(M)$, which is most commonly either the accuracy (to be maximized) or the error (to be minimized). Because \vec{x}_i and y_i are fixed, the ability of the model to generate predictions depends necessarily on θ (and perhaps also on random starting conditions in, for example, neural networks). Regardless of whether or not the model

parameters are discrete¹ or continuous², it is possible to fit a regression function through the score function values $\Psi(M_{\theta^*})$ at the point θ^* .

Using this architecture, the fundamental optimization procedure is as follows: **Algorithm**

1: Original Gaussian Process Optimization

Input: A labeled data set

$\{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\}$ and parameters $\theta_0 \in \Theta$.

Output: Proposed best parameters θ_{best} which maximize the score function.

Algorithm: Learn M_{θ_0} using the data and θ_0 .

Evaluate $\Psi(M_{\theta_0})$ and initialize set of tuples $\{(\theta_i, \Psi(M_{\theta_i}))\}$ with $(\theta_0, \Psi(M_{\theta_0}))$.

Initialize $\theta_{\text{best}} = \theta_0$.

While: Stopping criterion **False**

Fit a Gaussian process to $\{(\theta_i, \Psi(M_{\theta_i}))\} \forall i$. Infer a $\theta^* \in \Theta$ that is anticipated to yield the greatest difference $\Psi(M_{\theta^*}) - \Psi(M_{\theta_{\text{best}}})$ by an anticipation equation.

Evaluate $\Psi(M_{\theta^*})$ and add tuple $(\theta^*, \Psi(M_{\theta^*}))$ to $\{(\theta_i, \Psi(M_{\theta_i}))\}$.

If: $\Psi(M_{\theta^*}) > \Psi(M_{\theta_{\text{best}}})$
 $\theta_{\text{best}} = \theta^*$

Return: θ_{best}

The weakness of this algorithm is that, under most circumstances, if there is *no* indication that a scoring function evaluation at θ^* will lead to improvement, that point will not be evaluated. This is true even when the Gaussian process knows very little about the nature of the function at θ^* . As a result, this optimization procedure can be prone to finding poor local minima due to, for instance, bad initialization of θ_0 . This can be combatted to an extent by pursuing multiple random starts of the algorithm, however that process begins to resemble precisely the kind of cross-validation procedure we wished to avoid.

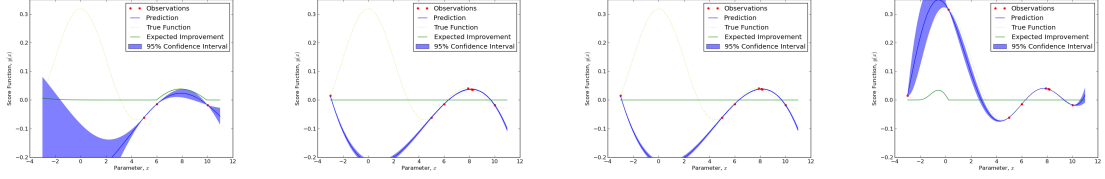
In the algorithm, we indicate an unspecified stopping criterion for the optimization. In our experiments, we specify that the algorithm should complete a predetermined number steps unless it converges to a maximum (either local or global) of its own accord and suspects that no further function evaluations are worthwhile, in which case termination is immediate.

3 A Hybrid Optimization Algorithm

It is apparent that it would be preferable if our optimization algorithm incorporated in itself a mech-

¹For example, consider the number of trees grown in a decision forest.

²For example, consider the σ^2 parameter in a SVM with a Gaussian kernel.



(a) Initial position for both the original and the Monte Carlo algorithms. (b) Original terminates at five iterations and discovers local maximum. (c) Monte Carlo variant finds local maximum identically to original. (d) Monte Carlo investigates high-uncertainty area and finds optimum.

Figure 1: Demonstration of the enhanced Monte Carlo-based algorithm versus the original expected improvement Gaussian process optimization procedure. Notice that the enhanced algorithm finds the global optimum of the score function, whereas the alternative does not.

anism to search for maxima in regions about which the Gaussian process can infer very little. However, it is equally apparent that an algorithm that searches only in those low-knowledge regions will be inefficient. Therefore, a superior algorithm would choose to evaluate regions of high uncertainty only a small fraction of the time, and would otherwise devote its attention to maximizing the scoring function in the typical fashion. To this end, we propose to incorporate what nearly amounts to a Metropolis-Hastings-like step such that the algorithm will use biased “coin flips” to determine whether or not an uncertain region is evaluated in the next iteration.

We note that exploring the region of highest uncertainty offers an immediate advantage over other common, uncertainty-based approaches, namely the method of searching the Gaussian process’ upper confidence bound. In particular, the upper confidence bound would require the additional tuning of a width parameter ω . We can begin to express this idea in the following algorithm, which preserves the core of the Gaussian process optimization algorithm, yet incorporates a kind of exploratory awareness that can lead to gains.

Algorithm 2: Hybrid Gaussian Process Optimization **Input:** Labeled data set

$\{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\}$ and parameters $\theta_0 \in \Theta$.

Output: Proposed best parameters θ_{best} which maximize the score function.

Algorithm: Learn M_{θ_0} on the data and θ_0 .

Evaluate $\Psi(M_{\theta_0})$ and initialize set of tuples $\{(\theta_i, \Psi(M_{\theta_i}))\}$ with $(\theta_0, \Psi(M_{\theta_0}))$.

Initialize $\theta_{\text{best}} = \theta_0$ and set a threshold $\tau \in [0, 1]$.

While: Stopping criterion **False**

Fit a Gaussian process to $\{(\theta_i, \Psi(M_{\theta_i}))\} \forall i$.

Infer a $\theta^* \in \Theta$ that is anticipated to yield the greatest difference $\Psi(M_{\theta^*}) - \Psi(M_{\theta_{\text{best}}})$ by an anticipation equation.

Obtain the closed-form standard deviations of all points in Θ and retrieve that point

$$\theta^u = \operatorname{argmax} \left(\sqrt{\sigma^2(\theta)} \right) \forall \theta \in \Theta.$$

Generate a random uniform value $\rho \in (0, 1)$.

If: $\rho < \tau$

Evaluate $\Psi(M_{\theta^*})$ and add tuple $(\theta^*, \Psi(M_{\theta^*}))$ to $\{(\theta_i, \Psi(M_{\theta_i}))\}$

If: $\Psi(M_{\theta^*}) > \Psi(M_{\theta_{\text{best}}})$

$$\theta_{\text{best}} = \theta^*$$

Else:

Evaluate $\Psi(M_{\theta^u})$ and add tuple $(\theta^u, \Psi(M_{\theta^u}))$ to $\{(\theta_i, \Psi(M_{\theta_i}))\}$

If: $\Psi(M_{\theta^u}) > \Psi(M_{\theta_{\text{best}}})$

$$\theta_{\text{best}} = \theta^u$$

Return: θ_{best}

In our experiments, we select the threshold $\tau = \frac{4}{5}$. In the interest of demonstrating the efficacy of our new algorithm, we construct a toy example that shows an instance where expected improvement optimization terminates before finding the global maximum, whereas our algorithm does precisely the opposite. In particular, for an input x , we define our score function by the equation, $y(x) = \frac{\sin(x)}{\pi x}$. We initialize both algorithms with an identical triplet of known function points, and ask the algorithms to run twenty iterations unless convergence is achieved. In the case of the original optimization algorithm, the Gaussian process quickly finds the local optimum, but chooses to discontinue searching the space after four iterations. By contrast, the hybrid architecture also finds the local optimum in four iterations, but then evaluates the point of largest uncertainty, which is near the global maximum. This phenomenon is illustrated in Figure 1 and leads us to validate the hypothesis that our algorithm is capable of finding improved maxima in optimization problems.

Details of Experiments for the Employed Data Set			
Domain	Raw Features	Response	Data Set Cardinality
Australian Credit Scoring	16	Desired credit approval of individuals based on characteristics	690

Table 1: Data set descriptions for the experiments used to validate the efficacy of the proposed algorithm. We summarize here the domain of the application, the input features to the algorithm, the response variable we wish to predict and the number of examples provided in the data.

3.1 Variable Threshold Algorithm

For some purposes it may be desirable not to use a fixed threshold τ for selecting a proportion of instances to search areas of high uncertainty. We therefore present an additional algorithm which incorporates a dynamic thresholding for choosing to explore low-knowledge regions. This methodology is principled in the sense that it employs the probability of improvement of the highest uncertainty point as a scaling parameter on a “basis” threshold τ' , which may equal unity if so desired. This permits exploration of unknown spaces a portion of the time (unlike the original algorithm), yet also recognizes that it can be advantageous to focus closely on maximizing expected improvement in a fashion that is inversely proportional to the probability of improvement at the location of highest uncertainty in Θ .

Algorithm 3: Variable Threshold Gaussian Process Optimization Input: A labeled data set

$\{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\}$ and parameters $\theta_0 \in \Theta$.

Output: Proposed best parameters θ_{best} which maximize the score function.

Algorithm: Learn M_{θ_0} on the data and θ_0 .

Evaluate $\Psi(M_{\theta_0})$ and initialize set of tuples $\{(\theta_i, \Psi(M_{\theta_i}))\}$ with $(\theta_0, \Psi(M_{\theta_0}))$.

Initialize $\theta_{\text{best}} = \theta_0$ and set a “basis” threshold $\tau \in [0, \infty)$.

While: Stopping criterion **False**

Fit a Gaussian process to $\{(\theta_i, \Psi(M_{\theta_i}))\} \forall i$.

Infer a $\theta^* \in \Theta$ that is anticipated to yield the greatest difference $\Psi(M_{\theta^*}) - \Psi(M_{\theta_{\text{best}}})$ by an anticipation equation.

Obtain the closed-form standard deviations of all points in Θ and retrieve that point

$$\theta^u = \operatorname{argmax} \left(\sqrt{\sigma^2(\theta)} \right) \forall \theta \in \Theta.$$

Obtain the probability of improvement for θ^u , ν and generate a random uniform value $\rho \in (0, 1)$.

If: $\rho < \nu\tau$

Evaluate $\Psi(M_{\theta^*})$ and add tuple $(\theta^*, \Psi(M_{\theta^*}))$ to $\{(\theta_i, \Psi(M_{\theta_i}))\}$

If: $\Psi(M_{\theta^*}) > \Psi(M_{\theta_{\text{best}}})$

$$\theta_{\text{best}} = \theta^*$$

Else:

Evaluate $\Psi(M_{\theta^u})$ and add tuple $(\theta^u, \Psi(M_{\theta^u}))$ to $\{(\theta_i, \Psi(M_{\theta_i}))\}$

If: $\Psi(M_{\theta^u}) > \Psi(M_{\theta_{\text{best}}})$

$$\theta_{\text{best}} = \theta^u$$

Return: θ_{best}

4 Experimental Results

We now turn our attention to analyzing the performance of the variable thresholding algorithm in application to a common machine learning benchmark. We use the Australian credit approval data set available at the UCI Machine Learning repository [4]. We summarize important statistics of this dataset in Table 1. We employ this dataset for testing the algorithm because it offers a range of variable types: continuous and categorical variables, in addition to missing values.

For the parameter selection stage, we train a random forest, which relies on minimizing the impurity measurement in a series of binary splits. To give an intuitive idea of the random forest’s approach to machine learning, we provide the following formal definition. Given a set of candidate splitting tests at a particular node in a decision tree, $S(\tau) = \{s_1^{(\tau)}, \dots, s_n^{(\tau)}\}$, we seek to split the data that satisfies:

$$s^* = \operatorname{argmax}_{s \in S(\tau)} - \sum_{c \in \mathcal{C}} P_c^{(\tau)} \log P_c^{(\tau)} \quad (9)$$

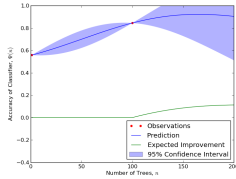
Where $P_c^{(\tau)}$ represents the class posterior probability (of class c) for the binary split for a data point located in the region of variable space identified as τ . In the case of this optimization experiment, we will attempt to identify the setting for the number of grown trees that simultaneously maximizes prediction accuracy and minimizes computation complexity.

For credit approval classification, our algorithm correctly identifies the optimal setting of parameters within ten iterations of the algorithm, having converged by the ninth. By contrast, the original Gaussian optimization algorithm fails to identify the best number of trees to create in the forest, opting for a value far larger than is empirically shown to be

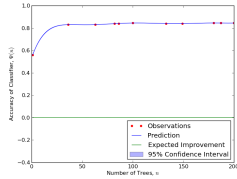
Details of Experiments for the Variable Threshold Algorithm				
<i>Statistic</i>	<i>Average</i>	<i>Minimum</i>	<i>Maximum</i>	<i>Standard Deviation</i>
Predictive Accuracy of Random Forest	85%	81%	90%	3.24%
Convergence Time of Optimization Algorithm	10	7	12	2.2

Table 2: We present here some relevant statistics related to our experiments in parameter optimization. Notice that in the predictive accuracy criterion, larger values are preferable. By contrast, we have that convergence time is better for smaller values. We define as convergence time the number of iterations of the algorithm that are required to map out completely the known behavior of the accuracy function.

necessary. The variable threshold process of parameter selection, by virtue of its exploratory capability, identifies that approximately forty decision trees are necessary to achieve maximum accuracy on unnormalized features. By contrast, the original approach terminates with a selection of 97 decision trees, a significant increase in the computation complexity of the learning algorithm. We report in Table 2. some of the statistics related to the classification results of the random forest and of the convergence of the variable threshold algorithm.



(c) Initial classification results for the original Gaussian process optimization procedure and the modified, variable thresholding approach. Notice that initially the inclusion of more trees is anticipated to improve the algorithm’s predictive performance.



(d) The final position achieved using variable thresholding. Notice that the algorithm has identified a low-uncertainty path the correctly predicts the nature of classification for all conceivable numbers of decision trees in the random forest.

5 Conclusion and Discussion

We have presented here two novel frameworks for Gaussian process optimization. In the case of variable thresholding, we find that we are able to produce results that are superior to those yielded by the original Gaussian process approach. We believe that this particular approach to hyper-parameter value assignment has many benefits over

other competing techniques such as k -fold cross-validation. We hope that these algorithms will find application in other machine learning applications where parameter optimization is crucial. In particular, we foresee applications to neural network learning as a future application of the approach.

References

- [1] Snook, Jasper and Hugo Larochelle and Ryan Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. arXiv. August 29, 2012.
- [2] Frean, Marcus and Phillip Boyle. *Using Gaussian Processes to Optimize Expensive Functions*. Victoria University of Wellington. 2008.
- [3] Benassi, Romain and Julien Bect and Emmanuel Vazquez. *Robust Gaussian Process-Based Global Optimization Using a Fully Bayesian Expected Improvement Criterion*. Learning and Intelligent Optimization. 2011.
- [4] Bache, K. & Lichman, M. *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science. 2013.
- [5] Compustat Database. *Wharton Research Data Services*. University of Pennsylvania, Web. 13 Apr. 2013. <https://wrds-web.wharton.upenn.edu/wrds/>.